Notice

Apple Computer, Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

Disclaimer of All Warranties and Liabilities

Apple Computer, Inc. makes no warranties, either express or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer, Inc. software is sold or licensed "as is." The entire risk as to its quality and performance is with the buyer. Should the programs prove defective following their purchase, the buyer (and not Apple Computer, Inc., its distributor, or its retailer) assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Apple Computer, Inc. be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software, even if Apple Computer, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This manual is copyrighted. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer, Inc.

© 1982 by Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

The word Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the U.S.A and Canada.

Written by Allen Watson of the Apple PCDP Publications Department

Warning

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

APPLE Product #A2L2007
Radio and Television Interference

The equipment described in this manual generates and uses radio-frequency energy. If it is not installed and used properly, that is, in strict accordance with our instructions, it may cause interference with radio and television reception.

This equipment has been tested and complies with the limits for a Class B computing device in accordance with the specifications in Subpart J, Part 15, of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation, especially if you use a “rabbit ear” television antenna. (A “rabbit ear” antenna is the telescoping-rod type usually contained on TV receivers.)

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripheral devices. To further isolate the problem:

- Disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it is caused by either the peripheral device or its I/O cable. These devices usually require shielded I/O cables. For Apple peripheral devices, you can obtain the proper shielded cable from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

If your computer does cause interference to radio or television reception, you can try to correct the interference by using one or more of the following measures:

- Turn the TV or radio antenna until the interference stops.
- Move the computer to one side or the other of the TV or radio.
- Move the computer farther away from the TV or radio.
- Plug the computer into an outlet that is on a different circuit than the TV or radio. (That is, make certain the computer and the radio or television set are on circuits controlled by different circuit breakers or fuses.)
- Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and TV.
If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find helpful the following booklet, prepared by the Federal Communications Commission:

“How to Identify and Resolve Radio-TV Interference Problems”

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, stock number 004-000-00345-4.
Table of Contents

Who Needs To Read This Supplement?

- vii Users: A Card Is a Card
- viii Developers: How To Use the Auxiliary Memory
- viii Contents of This Supplement
- ix Symbols Used in This Supplement

Introduction

- 3 Installation
- 4 80-Column Features
- 4 About the Auxiliary Memory

How the Auxiliary Memory Works

- 7 Addressing the Auxiliary Memory
- 9 How the 80-Column Display Works
- 11 Double High-Resolution Graphics

How To Use the Auxiliary Memory

- 16 The Extended Display
- 16 Display Pages
- 17 Display Mode Switching
- 18 Addressing the 80-Column Display Directly
- 21 Auxiliary Memory Switching
- 21 Switching the 48K Bank
- 25 Switching High Memory, Stack, and Zero-Page
- 29 Auxiliary-Memory Subroutines
- 29 Moving Data To Auxiliary Memory
- 30 Transferring Control To Auxiliary Memory
Programming Examples

35 Identifying Different Configurations
37 Apple Ile Identification in Assembly Language
40 Apple Ile Identification from BASIC
41 Apple Ile Identification from Pascal
43 Storing Graphics Pages from Applesoft
46 Storing Data Strings from Pascal

Index

Schematic Diagram

Extended Text Card Supplement
Who Needs To Read This Supplement?

This supplement comes with the Apple Ile Extended 80-Column Text Card and describes the added features it has, compared to the 80-Column Text Card. Before reading this supplement, you should read the Apple Ile 80-Column Text Card Manual.

There are two ways you are likely to use the extended version of the 80-Column Text Card:

- As a user with application programs that take advantage of the extra memory on the card to give you more features or more storage for your data.
- As a developer creating a program, for yourself or for others, that will use the extra storage the extended card provides.

Users: A Card Is a Card

From the user's point of view, the Extended 80-Column Text Card is just like the standard 80-Column Text Card. Oh, it's a little bigger, and it costs more, but the technical differences between the two kinds of text cards are mostly hidden by software. Read Chapter 1 of this supplement for an introduction to the Apple Ile 80-Column Extended Text Card.

The extended text card is installed the same way as the standard 80-column card: read the Apple Ile 80-Column Text Card Manual for directions.
Most application programs run the same with either card—in fact, many of them don’t even take advantage of the extra memory on the extended card; they simply use it to display 80 columns of text. Programs that do use the extra memory may do so automatically, without any action on your part, or they may let you select optional features or data storage. To find out how to use those programs with the extra memory, refer to their instruction manuals.

In short, if you just want to use this card for displaying 80 columns of text, and you aren’t developing a program that uses the auxiliary memory, all you really need to know can be found in the Apple IIe 80-Column Text Card Manual and in the instructions for your application programs.

**Developers: How To Use the Auxiliary Memory**

The only difference between the Extended 80-Column Text Card and the standard 80-Column Text Card is the amount of memory they contain. The extended card has 64K bytes of auxiliary memory, while the standard card has only the additional 1K bytes necessary to display 80 columns of text on an Apple IIe.

The main purpose of this supplement is to provide you with enough information to use the auxiliary memory in your programs. Normally, programs used with the Apple IIe can only work with the 64K bytes of built-in main memory. To work with the auxiliary memory, a program must set special switches in the Apple IIe that substitute auxiliary memory for main memory. Neither DOS 3.3 nor Pascal 1.1—system programs for the Apple II—support this memory substitution, so for now your application programs have to handle it themselves.

**Contents of This Supplement**

This supplement contains the information you need to use the auxiliary memory for storing programs and data. Chapter 1 is a general introduction; it describes the functions of the Extended 80-Column Text Card.

Chapter 2 is a general description of the design of the Extended 80-Column Text Card; it explains how the card works with the Apple IIe hardware.
Chapter 3 contains directions for using the auxiliary memory with your programs. Most of the information in Chapter 3 is adapted from the Apple IIe Reference Manual. The reference manual is your main source of information about the internal operation of the Apple IIe.

Chapter 4 contains short programs that use the auxiliary memory. These examples are functional, but not general: you will probably want to modify them for use in the programs you write.

**Symbols Used in This Supplement**

Special text in this manual is set off in different ways, as shown in these examples.

---

**Warning**

Important warnings appear in boxes like this.

---

**Reminder**:

Information that is only incidental to the text appears in gray boxes like this. You may want to skip over such boxes and return to them later.

Captions, definitions, and other short items appear in **marginal glosses** like this.
Introduction

3 Installation
4 80-Column Features
4 About the Auxiliary Memory
Introduction

The design of the Apple Ile Extended 80-Column Text Card is the same as that of the standard Apple Ile 80-Column Text Card. The only difference is that the extended text card contains 64K bytes of auxiliary memory (programmable memory or RAM) while the standard card contains only 1K byte of RAM. The 80-column display requires only 1K byte of auxiliary memory, so it will work with either card. The firmware that supports the special features associated with the 80-column display is part of the Apple Ile itself, and works the same regardless of which card is present.

Installation

Installing the Extended 80-Column Text Card is easy: do it just the way you install the standard 80-Column Text Card. Either card fits into the auxiliary slot (labeled AUX. CONNECTOR) on the main logic board inside the Apple Ile. If you haven’t installed the card yet, follow the directions given in the Apple Ile 80-Column Text Card Manual.

Warning

Never install or remove anything inside the Apple Ile with the power on. There is a small red lamp—an LED—toward the back of the main circuit board to remind you of this; if the red lamp is on, turn off the power before you do anything inside the Apple Ile.
80-Column Features

The built-in firmware that supports the 80-column display has other features in addition to the wider display. The Apple Ile 80-Column Text Card Manual tells you how to activate the built-in firmware and the 80-column display. That manual also describes many of the Apple Ile's features.

You can find more information about the Apple Ile in the Apple Ile Reference Manual. Chapter 2 includes a description of the different display modes and how to select them. Chapter 3 includes tables of the functions of the escape sequences and control keys in the Apple Ile.

About the Auxiliary Memory

The Extended 80-Column Text Card has 64K bytes of additional RAM, usually referred to as auxiliary memory. A 1K-byte area of this memory serves the same purpose as the memory on the 80-Column Text Card: expanding the text display to 80 columns. The other 63K bytes can be used for auxiliary program and data storage. If you use only 40 columns for text display, all 64K bytes are available for programs and data.

The processor in the Apple Ile can only address 64K bytes of memory. The computer has special circuits that programs can switch to access auxiliary memory in place of main memory. At any one time, locations in the same 64K address space are in either main memory or auxiliary memory. In other words, even though an Apple Ile with an Extended 80-Column Text Card has a total of 128K bytes of programmable memory, it is not appropriate to call it an 128K-byte system. Rather, there are 64K bytes of auxiliary memory that can be swapped for main memory under program control.

Warning

Careless switching to the auxiliary memory is almost certain to crash your programs. If you want to use auxiliary memory in your own programs, be sure to study the rest of this supplement and the relevant information in the Apple Ile Reference Manual.
How the Auxiliary Memory Works

7 Addressing the Auxiliary Memory
9 How the 80-Column Display Works
11 Double High-Resolution Graphics
How the Auxiliary Memory Works

This chapter briefly outlines how the auxiliary memory operates. It will help you understand what happens when you use the auxiliary memory in your programs.

Addressing the Auxiliary Memory

The 6502 microprocessor can address 64K bytes of memory. In the Apple IIe the microprocessor’s entire 64K memory space is taken up by main RAM (random-access memory), ROM (read-only memory), and I/O (input/output); there’s no memory space available for the added memory on the extended text card. Instead, the address bus is connected to the auxiliary memory in parallel with the main memory. To use the auxiliary memory for program and data storage, the Apple IIe switches its data bus so that it reads and writes to the memory on the card instead of the main memory. To use the auxiliary memory to expand the display, the Apple IIe fetches data both from main memory and from auxiliary memory, as described in the section “How the 80-Column Display Works.”

The bus switching for program and data storage is controlled by the Memory Management Unit (MMU), a custom integrated circuit designed for the Apple IIe (see Chapter 7 of the Apple IIe Reference Manual). The MMU contains the soft switches set by your programs along with the logic circuitry to monitor the address bus and to switch to auxiliary memory for the selected address ranges.
Figure 2-1. Memory Map with Auxiliary Memory

<table>
<thead>
<tr>
<th>Memory Offset</th>
<th>Main Memory</th>
<th>Auxiliary Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E000</td>
<td>Bank-Switched Memory</td>
<td>Bank-Switched Memory</td>
</tr>
<tr>
<td>$D000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$9000</td>
<td></td>
<td>Hi-Res Graphics Page 2</td>
</tr>
<tr>
<td>$8000</td>
<td>Hi-Res Graphics Page 1</td>
<td>Hi-Res Graphics Page 1X</td>
</tr>
<tr>
<td>$7000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6000</td>
<td></td>
<td>Hi-Res Graphics Page 2</td>
</tr>
<tr>
<td>$5000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000</td>
<td></td>
<td>Hi-Res Graphics Page 1</td>
</tr>
<tr>
<td>$3000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2000</td>
<td></td>
<td>Text Page 1X</td>
</tr>
<tr>
<td>$1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$000</td>
<td></td>
<td>Text Page 2</td>
</tr>
<tr>
<td>$FFF</td>
<td>Stack &amp; Zero Page</td>
<td>Stack &amp; Zero Page</td>
</tr>
<tr>
<td>$0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Extended Text Card Supplement
As you can see by studying the memory map in Figure 2-1, the auxiliary memory is divided into two large sections and one small one. The largest section is substituted for main memory addresses 512 to 49151 ($200 through $BFFF). This part of memory is sometimes referred to as the 48K memory space, and it is used for storing programs and data.

The other large section of auxiliary memory replaces main memory addresses 52K to 64K ($D000 through $FFFF). This memory space is called the bank-switched memory. If you plan to use this part of the auxiliary memory, read the section "Bank-switched Memory" in the Apple IIe Reference Manual. The switching for the ROM and the $0000 bank is independent of the auxiliary-RAM switching, so the bank switches have the same effect on the auxiliary RAM that they do on the main RAM.

When you switch to the auxiliary memory in the bank-switched memory space, you also get the first two pages of auxiliary memory, from 0 to 511 ($0000 through $01FF). This part of memory contains page zero, which is used for important data and base addresses, and page one, which is the 6502 stack.

**Warning**

Remember that addresses in page zero and the 6502 stack switch to auxiliary memory any time you switch the bank-switched memory to auxiliary memory.

**How the 80-Column Display Works**

Half of the data for the 80-column display is stored in main memory in the normal text Page 1, and the other half is stored in auxiliary memory on the extended text card. The display circuitry fetches bytes of data from these two memory areas simultaneously and displays them as two adjacent characters.

The main memory and the auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The 40-column display uses every other clock cycle and fetches data only from main memory. The 80-column display uses the remaining clock cycles to process the additional display data from auxiliary memory.
The byte of display data from main memory goes to a buffer on the main logic board, and the display data from auxiliary memory goes to a buffer on the extended text card. When the 80-column display is on, the data bytes from these buffers are switched onto the video data bus on alternate clock cycles: first the byte from the auxiliary memory, then the byte from the main memory. The main memory provides the characters displayed in the odd columns of the display, and the auxiliary memory provides the characters in the even columns.

The 80-column display contains twice as many characters as the 40-column display does, so it has to put twice as many dots across the screen. This means that the dots are clocked out at 14MHz instead of 7MHz, making them narrower and therefore dimmer on a normal video monitor. On a television set, the dot patterns making up the characters are too close together to reproduce clearly. To produce a satisfactory 80-column display requires a monitor with a bandwidth of at least 14MHz.
RGB stands for red, green, and blue, and identifies a type of color monitor that uses independent inputs for the three primary colors.

Except for some expensive RGB-type color monitors, any video monitor with a bandwidth as high as 14MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

Note that this simultaneous-then-sequential fetching applies only to the video-display generation; reading and writing for data storage in auxiliary memory is done by switching the data bus to read only from the card, as described in the previous section. For more information about the way the Apple IIe handles its display memory, refer to Chapter 2 and Chapter 7 of the Apple IIe Reference Manual.

**Double High-Resolution Graphics**

When you select mixed-mode graphics with 80-column text, you would expect that the doubling of the data rate that produces the 80-column display would change the high-resolution graphics from 280 to 560 dots horizontally and cause the low-resolution graphics to malfunction. To prevent this, the logic that controls the display includes an extra circuit to force the graphics displays to be the same regardless of whether you have set the soft switches for 80-column text or for 40-column text. This feature is included so that you can use 80-column text in the mixed graphics and text modes.

For those who would like to have a graphics display with twice the horizontal resolution, there is a way to disable the circuit that forces normal graphics timing with 80-column text. There are two things you must do to obtain the double high-resolution display:

- Install a jumper to connect the two Molex-type pins on the Extended 80-Column Text Card.
- Turn on the Annunciator 3 soft switch along with the switches that select the 80-column display and high-resolution graphics.

This procedure works only on the Apple IIe with the Rev B (and later) main logic board, identified by a B as the last letter of the part number on the back part of the board. Connecting the pins on the Extended 80-Column Text Card completes a connection between pin 50 (AN3) and pin 55 (FRCTXT') on the auxiliary slot.

**How the Auxiliary Memory Works**
Warning
If you have a Rev A Apple IIe, using an extended text card with a jumper makes the computer inoperable. You cannot use the double high-resolution modification with a Rev A Apple IIe.

If you have an extended text card with a jumper installed in a Rev B (or later) Apple IIe, turning on Annunciator 3 and selecting high-resolution graphics and 80-column text at the same time generates a display using high-resolution Page 1 addresses in main memory and auxiliary memory at the same time.

The memory mapping for this graphics display is doubled by columns the same way as 80-column text, but it uses high-resolution graphics Page 1 instead of text Page 1. Where the 80-column text mode displays pairs of data bytes as pairs of characters, double high-resolution mode displays pairs of data bytes as 14 adjacent dots, seven from each byte. As in 80-column text mode, there are twice as many dots across the display screen, so the dots are only half as wide.

Existing Apple II graphics programs do not support this kind of display. Until new programs become available, you'll have to write your own plotting routines if you want to use 560-dot graphics.
Chapter 3

How To Use the Auxiliary Memory

16 The Extended Display
16 Display Pages
17 Display Mode Switching
18 Addressing the 80-Column Display Directly
21 Auxiliary Memory Switching
21 Switching the 48K Bank
25 Switching High Memory, Stack, and Zero-Page
29 Auxiliary-Memory Subroutines
29 Moving Data To Auxiliary Memory
30 Transferring Control To Auxiliary Memory
Chapter 3

How To Use the Auxiliary Memory

This chapter describes soft switches and built-in subroutines that control the operation of the auxiliary memory. To take advantage of the additional memory, you must set up your programs to operate in one part of memory while they switch the other part between main and auxiliary RAM. Your program can perform the memory switching by means of the soft switches described in the section “Display Mode Switching” or by using the AUXMOV and XFER subroutines described later in this chapter. Except for these subroutines, most existing Apple II system software (DOS 3.3, Pascal 1.1) doesn’t support the auxiliary memory.

Although some high-level languages, such as BASIC, can set the soft switches directly, your programs must use assembly-language subroutines to control the auxiliary memory. Small assembly-language subroutines can be accessed from a BASIC program using a CALL statement, or they can be linked to a Pascal program as procedures or functions: see the examples in Chapter 4.

Warning
Do not attempt to use the auxiliary memory directly from a program in an interpreter language such as BASIC or Pascal. The interpreters that run such programs use several areas in main memory, including the stack and the zero page. If you switch to auxiliary memory in these pages, the interpreter crashes. When you reset the system to start over, your program and data are lost.
The Extended Display

The primary purpose of an 80-column text card is the generation of an 80-column display, so there is a complete set of switches just to control the display. Other switches are used for program and data storage in the auxiliary memory; they are described later.

Display Pages

The Apple IIe generates its video displays from data stored in specific areas in memory called display pages. The 40-column-text and low-resolution-graphics modes use text Page 1 and text Page 2, located at 1024-2047 (hexadecimal $400-$7FF) and 2048-3071 ($800-$BFF) in main memory.

The 80-column text display uses a combination of text Page 1 in main memory and the same page in the auxiliary memory, here called Page 1X. Text Page 1X occupies the same address space as text Page 1, but in auxiliary memory rather than main memory. To store data in Page 1X, you must use a soft switch (see the section “Display Mode Switching”). The built-in 80-column display routines described in Chapter 3 of the Apple IIe Reference Manual take care of this switching automatically; that is a good reason to use those routines for all your normal 80-column text output.

<table>
<thead>
<tr>
<th>Display Mode</th>
<th>Page</th>
<th>Lowest Address</th>
<th>Highest Address</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>40-Column Text Low-Resolution Graphics</td>
<td>1</td>
<td>$400 1024</td>
<td>$7FF 2047</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$800 2048</td>
<td>$BFF 3071</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80-Column Text</td>
<td>1</td>
<td>$400 1024</td>
<td>$7FF 2047</td>
<td></td>
</tr>
<tr>
<td>Normal 280-Dot High-Resolution Graphics</td>
<td>1</td>
<td>$2000 8192</td>
<td>$3FF 16383</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$4000 16384</td>
<td>$5FF 24575</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional 560-Dot High-Resolution Graphics</td>
<td>1</td>
<td>$2000 8192</td>
<td>$3FF 16383</td>
<td></td>
</tr>
</tbody>
</table>
Display Mode Switching

You select the display mode that is appropriate for your application by reading or writing to soft switches. Most soft switches have three memory locations: one for turning the switch on, one for turning it off, and one for reading the state of the switch.

Table 3-2 shows the locations of the soft switches that control the display modes. The table gives the switch locations in three forms: hexadecimal, decimal, and negative decimal. You can use the hexadecimal values in your machine-language programs. Use the decimal values in PEEK or POKE commands in Applesoft BASIC; the negative values are for Integer BASIC.

Some of the soft switches in Table 3-2 are marked read or write. Those soft switches share their locations with the keyboard data and strobe functions. To perform the function shown in the table, use only the operation listed there. Soft switches that are not marked may be accessed by either a read or a write. When writing to a soft switch, it doesn't matter what value you write; the switch function occurs when you address the location, and the value is ignored.

Warning

Be sure to use only the indicated operations to manipulate the switches. If you read from a switch marked write, you won't get the correct data. If you write to a switch marked read, you won't set the switch you wanted, and you may change some other switch so as to cause your program to malfunction.

When you read a soft switch, you get a byte with the state of the switch in bit 7, the high-order bit. The other bits in the byte are unpredictable. If you are programming in machine language, this bit is the sign bit. If you read a soft-switch from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if the switch is on, the value will be equal to or greater than 128; if the switch is off, the value will be less than 128.
Table 3-2. Display Soft Switches. (1) This mode is only effective when TEXT switch is off. (2) This switch has a different function when 80 STORE is on: refer to the next section. (3) This switch changes the function of the PAGE2 switch for addressing the display memory on the extended text card: refer to the next section.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Location</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>On: Display Text</td>
<td>$c051 49233 -16303</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off: Display Graphics</td>
<td>$c050 49232 -16304</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read TEXT Switch</td>
<td>$c01a 49178 -16358</td>
<td>Read</td>
</tr>
<tr>
<td>MIXED</td>
<td>On: Text With Graphics</td>
<td>$c053 49235 -16301</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Off: Full Graphics</td>
<td>$c052 49234 -16302</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Read MIXED Switch</td>
<td>$c018 49179 -16357</td>
<td>Read</td>
</tr>
<tr>
<td>PAGE2</td>
<td>On: Display Page 2</td>
<td>$c055 49237 -16299</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Off: Display Page 1</td>
<td>$c054 49236 -16300</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Read PAGE2 Switch</td>
<td>$c01c 49180 -16356</td>
<td>Read</td>
</tr>
<tr>
<td>HIRES</td>
<td>On: Graphics = High-Resolution</td>
<td>$c057 49239 -16297</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Off: Graphics = Low-Resolution</td>
<td>$c056 49238 -16298</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Read HIRES Switch</td>
<td>$c01d 49181 -16355</td>
<td>Read</td>
</tr>
<tr>
<td>80COL</td>
<td>On: Display 80 Columns</td>
<td>$c000 49165 -16371</td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td>Off: Display 40 Columns</td>
<td>$c00c 49164 -16372</td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td>Read 80COL Switch</td>
<td>$c01f 49183 -16353</td>
<td>Read</td>
</tr>
<tr>
<td>80 STORE</td>
<td>On: Store in Auxiliary Page</td>
<td>$c001 49153 -16383</td>
<td>Write,3</td>
</tr>
<tr>
<td></td>
<td>Off: Store in Main Page</td>
<td>$c000 49152 -16384</td>
<td>Write,3</td>
</tr>
<tr>
<td></td>
<td>Read 80 STORE Switch</td>
<td>$c018 49176 -16360</td>
<td>Read</td>
</tr>
</tbody>
</table>

Addressing the 80-Column Display Directly

Figure 3-1 is the map of the 80-column display. Half of the data is stored in text Page 1 in main memory, and the other half is stored in the same locations in auxiliary memory (here called Page 1X). The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the auxiliary memory stores the characters in the even columns. For a full description of the way the Apple IIe handles its display memory, refer to Chapter 2 and Chapter 7 of the Apple IIe Reference Manual.

Extended Text Card Supplement
To store data directly into the display page on the Extended 80-Column Text Card, first turn on the 8STORE soft switch by writing to location 49153 (negative decimal -16383 or hexadecimal $C001). With 8STORE on, the page-select switch PAGE2 switches between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in Page 1X in auxiliary memory. To select Page 1X, turn the PAGE2 soft switch on by reading or writing at location 49237 (-16299, $C055).

You'll have to write a short program to try out the 8STORE and PAGE2 soft switches. When you try to change these switches by using the Monitor program, it changes them back in the process of displaying the commands you type.

If you want to use the optional double-high-resolution display described in Chapter 2, you can store data directly into high-resolution graphics Page 1X in auxiliary memory in a similar fashion. Turn on both 8STORE and HIRES, then use PAGE2 to switch from Page 1 in main memory to Page 1X in auxiliary memory.

The memory mapping for double high-resolution graphics is similar to the normal high-resolution mapping described in Chapter 2 of the Apple IIe Reference Manual, with the addition of the column doubling produced by the 80-column display. Like the 80-column text mode, the double high-resolution graphics mode displays two bytes in the time normally required for one, but it uses high-resolution graphics Page 1 and Page 1X instead of text Page 1 and Page 1X.

Double high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, seven from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appears in columns 0-6, 14-20, etc., up to columns 547-552. Data from main memory appears in columns 7-13, 21-27, and so on up to 553-559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth monitor, single dots will be dimmer than normal.

For a description of the way the high-order bit acts as color-select bit in high-resolution displays, see Chapters 2 and 7 of the Apple IIe Reference Manual.
Figure 3-1. Map of 80-Column Text Display

<table>
<thead>
<tr>
<th>MAIN MEMORY</th>
<th>$00</th>
<th>$01</th>
<th>$02</th>
<th>$03</th>
<th>$04</th>
<th>$05</th>
<th>$06</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUXILIARY MEMORY</td>
<td>$00</td>
<td>$01</td>
<td>$02</td>
<td>$03</td>
<td>$04</td>
<td>$05</td>
<td>$06</td>
</tr>
<tr>
<td>$400</td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$480</td>
<td>1152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$500</td>
<td>1280</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$580</td>
<td>1408</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$600</td>
<td>1536</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$680</td>
<td>1664</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$700</td>
<td>1792</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$780</td>
<td>1920</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$828</td>
<td>1064</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4A8</td>
<td>1192</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$528</td>
<td>1320</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5A8</td>
<td>1448</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$628</td>
<td>1576</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6A8</td>
<td>1704</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$728</td>
<td>1832</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7A8</td>
<td>1960</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$450</td>
<td>1104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4D0</td>
<td>1232</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$550</td>
<td>1360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5D0</td>
<td>1488</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$650</td>
<td>1616</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6D0</td>
<td>1744</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$750</td>
<td>1872</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7D0</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Auxiliary Memory Switching

This section describes the switches used to access the auxiliary memory for storing programs and data.

Warning

The display soft switches STORE, PAGE2, and HIRES, discussed here and in the previous section, are used primarily for addressing display data. These switches override the general-purpose switches described in this section, so you must set them correctly even if your program doesn’t use them.

Switching the 48K Bank

Switching the 48K-byte section of memory is performed by two soft switches: RAMRD selects main or auxiliary memory for reading, and RAMWRT selects main or auxiliary memory for writing. As shown in Table 3-3, each switch has a pair of memory locations dedicated to it, one to select main memory, and the other to select auxiliary memory. Setting the read and write functions independently makes it possible for a program whose instructions are being fetched from one 48K-byte memory space to store data into the other 48K memory space.

Warning

Before using these switches, you must fully understand the effects of switching to auxiliary memory. For example, an application program running in the 48K bank of auxiliary memory that tries to use the built-in I/O routines by calling the standard I/O links will crash even though the main ROM, which contains the built-in I/O routines, has been selected. This happens because the standard links call DOS routines, and DOS is in the 48K bank of main memory, which is locked out while the application program is running in auxiliary memory.

Writing to the soft-switch at location $C003 turns RAMRD on and enables auxiliary memory for reading; writing to location $C002 turns RAMRD off and enables main memory for reading. Writing to the soft-switch at location $C005 turns RAMWRT on and enables the auxiliary memory for writing; writing to location $C004 turns RAMWRT off and enables main memory for writing. By setting these switches independently, you can use any of the four combinations of reading and writing in main or auxiliary memory.

How To Use the Auxiliary Memory
Auxiliary memory corresponding to text Page 1 and high-resolution graphics Page 1 can be used as part of the 48K bank by using RAMRD and RAMWRT. These areas in auxiliary memory can also be controlled separately by using the display-page switches 80STORE, PAGE2, and HIRES described in "Addressing the 80-Column Display Directly."

As shown in Table 3-3, the 80STORE switch functions as an enabling switch: with it on, the PAGE2 switch selects main memory or auxiliary memory. With the HIRES switch off, the PAGE2 switch selects main or auxiliary memory in the text display Page 1, $0400 to $07FF; with HIRES on, the PAGE2 switch selects main or auxiliary memory in text Page 1 and high-resolution graphics Page 1, $2000 to $3FFF.

If you are using both the 48K-bank control switches and the display-page control switches, the display-page control switches take priority: if 80STORE is off, RAMRD and RAMWRT work for the entire memory space from $0200 to $FBFF, but if 80STORE is on, RAMRD and RAMWRT have no effect on the display page. Specifically, if 80STORE is on and HIRES is off, PAGE2 controls text Page 1 regardless of the settings of RAMRD and RAMWRT. Likewise, if 80STORE and HIRES are both on, PAGE2 controls both text Page 1 and high-resolution graphics Page 1, again regardless of RAMRD and RAMWRT.

You can find out the settings of these soft switches by reading from two other locations. The byte you read at location $C013 has its high bit (the sign bit) set to 1 if RAMRD is on (auxiliary memory is enabled for reading), or 0 if RAMRD is off (the 48K block of main memory is enabled for reading). The byte at location $C014 has its high bit set to 1 if RAMWRT is on (auxiliary memory is enabled for writing), or 0 if RAMWRT is off (the 48K block of main memory is enabled for writing).
Figure 3-2. Effect of Switching RAMRD and RAMWRT with BOSTORE Off

<table>
<thead>
<tr>
<th>Main Memory</th>
<th>Auxiliary Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF</td>
<td>Bank-Switched Memory</td>
</tr>
<tr>
<td>$DFFF</td>
<td></td>
</tr>
<tr>
<td>$D000</td>
<td></td>
</tr>
<tr>
<td>$CFFF</td>
<td></td>
</tr>
<tr>
<td>$6000</td>
<td>Hi-Res Graphics Page 2</td>
</tr>
<tr>
<td>$4000</td>
<td>Hi-Res Graphics Page 1</td>
</tr>
<tr>
<td>$2000</td>
<td></td>
</tr>
<tr>
<td>$C000</td>
<td>Text Page 2</td>
</tr>
<tr>
<td>$8000</td>
<td>Text Page 1</td>
</tr>
<tr>
<td>$4000</td>
<td></td>
</tr>
<tr>
<td>$2000</td>
<td></td>
</tr>
<tr>
<td>$1FF</td>
<td>Stack &amp; Zero Page</td>
</tr>
<tr>
<td>$0</td>
<td>Stack &amp; Zero Page</td>
</tr>
</tbody>
</table>

Active □ Inactive □ Switching □

RAMRD: X  RAMWRT: X  BOSTORE: off
PAGE2: off  HIRES: off  ALTZP: off

How To Use the Auxiliary Memory
Figure 3-3. Effect of Switching RAMRD and RAMWRT with BSTORE and HIRES On

<table>
<thead>
<tr>
<th>$FFFF</th>
<th>Main Memory</th>
<th>Auxiliary Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank-Switched Memory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $DFFF | |
| $0000 | |

| $BFFF | |

| $6000 | Hi-Res Graphics Page 2 |
| $4000 | Hi-Res Graphics Page 1 |
| $2000 | Hi-Res Graphics Page 1X |

| $000 | Text Page 2 |
| $0000 | Text Page 1 |
| $800 | Text Page 1X |

| $1FF | Stack & Zero Page |
| $0 | |

Active | Inactive | Switching |
---|---|---|
RAMRD: X | RAMWRT: X | BSTORE: on |
PAGE2: off | HIRES: on | ALTZP: off |

Extended Text Card Supplement
Switching High Memory, Stack, and Zero Page

The single soft switch ALTZP (alternate zero page) switches the bank-switched memory and the associated stack and zero page area between main and auxiliary memory. As shown in Table 3-3, writing to location $C009 turns ALTZP on and selects auxiliary-memory stack and zero page; writing to the soft switch at location $C008 turns ALTZP off and selects main-memory stack and zero page for reading and writing. The section “Auxiliary-Memory Subroutines” describes firmware that you can call to help you switch between main and auxiliary memory.

To find out the setting of this soft switch, read location $C016. The data byte you get has its high bit (the sign bit) set to 1 if ALTZP is on (the bank-switched area, stack, and zero page in the auxiliary memory are selected), or 0 if ALTZP is off (the same areas in main memory are selected).

To have enough memory locations for all of the soft switches and remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 3-3 share their memory locations with the keyboard functions listed in Chapter 2 of the *Apple IIe Reference Manual*. Whichever operation—read or write—is shown in Table 3-3 for controlling the auxiliary memory is the one that is not used for reading the keyboard and clearing the strobe.
Table 3-3. Auxiliary-Memory Select Switches. (1) When 80STORE is on, the PAGE2 switch works as shown; when 80STORE is off, PAGE2 doesn't affect the auxiliary memory. (2) When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to select between high-resolution Page 1 areas in main and auxiliary memory.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Location Hex</th>
<th>Location Decimal</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>80RDO</td>
<td>On: Read Aux. 48K</td>
<td>$C003</td>
<td>49155</td>
<td>-16381</td>
</tr>
<tr>
<td></td>
<td>Off: Read Main 48K</td>
<td>$C002</td>
<td>49154</td>
<td>-16382</td>
</tr>
<tr>
<td></td>
<td>Read 80RDO Switch</td>
<td>$C013</td>
<td>49171</td>
<td>-16365</td>
</tr>
<tr>
<td>80WRT</td>
<td>On: Write Aux. 48K</td>
<td>$C005</td>
<td>49157</td>
<td>-16379</td>
</tr>
<tr>
<td></td>
<td>Off: Write Main 48K</td>
<td>$C004</td>
<td>49156</td>
<td>-16380</td>
</tr>
<tr>
<td></td>
<td>Read 80WRT Switch</td>
<td>$C014</td>
<td>49172</td>
<td>-16354</td>
</tr>
<tr>
<td>ALTZP</td>
<td>On: Aux. Stack, Zero Page, and Bank-Switched Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off: Main Stack, Zero Page, and Bank-Switched Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read ALTZP Switch</td>
<td>$C009</td>
<td>49161</td>
<td>-16373</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C008</td>
<td>49160</td>
<td>-16374</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C016</td>
<td>49174</td>
<td>-16352</td>
</tr>
<tr>
<td>80STORE</td>
<td>On: Access Page 1X</td>
<td>$C001</td>
<td>49153</td>
<td>-16383</td>
</tr>
<tr>
<td></td>
<td>Off: Use 80RDO, 80WRT</td>
<td>$C000</td>
<td>49152</td>
<td>-16384</td>
</tr>
<tr>
<td></td>
<td>Read 80STORE Switch</td>
<td>$C018</td>
<td>49176</td>
<td>-16360</td>
</tr>
<tr>
<td>PAGE2</td>
<td>On: Access Aux. Memory</td>
<td>$C055</td>
<td>49237</td>
<td>-16299</td>
</tr>
<tr>
<td></td>
<td>Off: Access Main Memory</td>
<td>$C054</td>
<td>49236</td>
<td>-16300</td>
</tr>
<tr>
<td></td>
<td>Read PAGE2 Switch</td>
<td>$C01C</td>
<td>49180</td>
<td>-16356</td>
</tr>
<tr>
<td>HIRES</td>
<td>On: Access High-Resolution Page IX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off: Use 80RDO, 80WRT</td>
<td>$C057</td>
<td>49239</td>
<td>-16297</td>
</tr>
<tr>
<td></td>
<td>Read HIRES Switch</td>
<td>$C010</td>
<td>49181</td>
<td>-16355</td>
</tr>
</tbody>
</table>
Figure 3-4. Effect of Switching ALTZP

<table>
<thead>
<tr>
<th>$FFFF</th>
<th>Main Memory</th>
<th>Auxiliary Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bank-Switched Memory</td>
<td>Bank-Switched Memory</td>
</tr>
<tr>
<td>$DFFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $8FFF |              |                 |

| $6000 | Hi-Res Graphics Page 2 |               |
| $4000 | Hi-Res Graphics Page 1  | Hi-Res Graphics Page 1X |
| $2000 |              |                 |

| $C00  | Text Page 2   |               |
| $800  | Text Page 1   | Text Page 1X  |
| $400  |              |                 |

| $200  |              |                 |

| $1FF  | Stack & Zero Page | Stack & Zero Page |
| $0    |                |                 |

<table>
<thead>
<tr>
<th>Active</th>
<th>Inactive</th>
<th>Switching</th>
</tr>
</thead>
</table>

RAWRD: off  RAWWRT: off  BSTORE: off
PAGE2: off  HIRES: off  ALTZP: X

How To Use the Auxiliary Memory
Figure 3-5. Effect of Switching PAGE2 with 80STORE and HIRES On

<table>
<thead>
<tr>
<th>Main Memory</th>
<th>Auxiliary Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF</td>
<td>Bank-Switched Memory</td>
</tr>
<tr>
<td>$0FFF</td>
<td>Bank-Switched Memory</td>
</tr>
<tr>
<td>$0000</td>
<td></td>
</tr>
<tr>
<td>$0000</td>
<td></td>
</tr>
<tr>
<td>$8000</td>
<td>Hi-Res Graphics Page 2</td>
</tr>
<tr>
<td>$4000</td>
<td>Hi-Res Graphics Page 1</td>
</tr>
<tr>
<td>$2000</td>
<td>Hi-Res Graphics Page 1X</td>
</tr>
<tr>
<td>$C000</td>
<td>Text Page 2</td>
</tr>
<tr>
<td>$8000</td>
<td>Text Page 2</td>
</tr>
<tr>
<td>$4000</td>
<td>Text Page 1</td>
</tr>
<tr>
<td>$2000</td>
<td>Text Page 1X</td>
</tr>
<tr>
<td>$1FF</td>
<td>Stack &amp; Zero Page</td>
</tr>
<tr>
<td>$0</td>
<td></td>
</tr>
</tbody>
</table>

Active □ Inactive □ Switching □

RAMRD: off RAMWR: off 80STORE: on

PAGE2: X HIRES: on ALTZP: off

Extended Text Card Supplement
Auxiliary-Memory Subroutines

If you want to write assembly-language programs or procedures that use auxiliary memory, the built-in auxiliary-memory subroutines will be helpful. These subroutines make it possible to use the auxiliary memory without having to manipulate the soft switches already described.

The subroutines described in this section make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid inexplicable crashes.

You use these built-in subroutines the same way you use the I/O subroutines described in Chapter 3 of the *Apple IIe Reference Manual*: by making subroutine calls to their starting locations. Those locations are shown in Table 3-4.

<table>
<thead>
<tr>
<th>Subroutine Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUXMOVE</td>
<td>$C311</td>
<td>Moves data blocks between main and auxiliary memory</td>
</tr>
<tr>
<td>XFER</td>
<td>$C314</td>
<td>Transfers program control between main and auxiliary memory</td>
</tr>
</tbody>
</table>

Moving Data To Auxiliary Memory

In your assembly-language programs, you can use the built-in subroutine named AUXMOVE to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page zero and set the carry bit to select the direction of the move—main to auxiliary or auxiliary to main.

⚠️ Warning

Don’t try to use AUXMOVE to copy data in page zero, page one (the 6502 stack), or in the bank-switched memory ($0000-$FFFF). AUXMOVE uses page zero while it is copying, so it can’t handle moves in the memory space switched by ALTZP.

Table 3-4. Auxiliary-Memory Routines

The carry bit is bit 0 in the processor status word; use the SEC instruction to set it, and CLC to clear it.
Remember that Pascal uses page zero too, so you can't use A4M0V0 from a Pascal procedure without saving the contents of page zero first, and restoring them afterward.

The pairs of bytes you use for passing addresses to this subroutine are called A1, A2, and A4; they are used for passing parameters to several of the Apple IIe's built-in routines. The addresses of these byte pairs are shown in Table 3-5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Parameter Passed</th>
</tr>
</thead>
</table>
| Carry | 1 = Move from main to auxiliary memory  
       0 = Move from auxiliary to main memory |
| A1L  | $3C      | Source starting address, low-order byte |
| A1H  | $3D      | Source starting address, high-order byte |
| A2L  | $3E      | Source ending address, low-order byte  |
| A2H  | $3F      | Source ending address, high-order byte  |
| A4L  | $42      | Destination starting address, low-order byte |
| A4H  | $43      | Destination starting address, high-order byte |

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

The A4M0V0 routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit (SEC); to copy data from auxiliary memory to main memory, clear the carry bit (CC). When you make the subroutine call to A4M0V0, the subroutine copies the block of data as specified by the A registers and the carry bit. When it is finished, the accumulator and the x and y registers are just as they were when you called it.

**Transferring Control To Auxiliary Memory**

You can use the built-in routine named XFER to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFER: the address of the routine you are transferring to, the direction of the transfer (main to auxiliary or auxiliary to main), and which page zero and stack you want to use.
### Table 3-6. Parameters for XFER Routine

<table>
<thead>
<tr>
<th>Name or Location</th>
<th>Parameter Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carry</td>
<td>1 = Transfer from main to auxiliary memory</td>
</tr>
<tr>
<td></td>
<td>0 = Transfer from auxiliary to main memory</td>
</tr>
<tr>
<td>Overflow</td>
<td>1 = Use page zero and stack in auxiliary memory</td>
</tr>
<tr>
<td></td>
<td>0 = Use page zero and stack in main memory</td>
</tr>
<tr>
<td>$3\text{ED}$</td>
<td>Program starting address, low-order byte</td>
</tr>
<tr>
<td>$3\text{EE}$</td>
<td>Program starting address, high-order byte</td>
</tr>
</tbody>
</table>

The overflow bit is bit 6 in the processor status word; use the CLV instruction to clear it. To set it, force an overflow by adding two numbers that total more than 127.

Put the transfer address into the two bytes at locations $3\text{ED}$ and $3\text{EE}$, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory. Use the overflow bit to select which page zero and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit to use the auxiliary memory.

### Warning

It is the programmer's responsibility to save the current stack pointer somewhere in the current memory space before using XFER and to restore it after regaining control. Failure to do so will cause program errors.

After you have set up the parameters, pass control to the XFER routine by a jump instruction, rather than a subroutine call. XFER saves the accumulator and the transfer address on the current stack, then sets up the soft switches for the parameters you have selected and jumps to the new program.
35 Identifying Different Configurations
37 Apple Ile Identification in Assembly Language
40 Apple Ile Identification from BASIC
41 Apple Ile Identification from Pascal
43 Storing Graphics Pages from Applesoft
46 Storing Data Strings from Pascal
Chapter 4

Programming Examples

This chapter contains examples showing how to use the auxiliary memory from a program. These examples are not intended to be universal routines that everyone can use as is; rather, they are representative examples showing how specific operations have been implemented. You will probably want to study the examples to see how it is done, then copy or modify them to suit your application.

Identifying Different Configurations

By identifying the configuration of the machine they are running on, application programs for the Apple IIe can take advantage of the new features and still remain compatible with older Apple II's. This section gives a procedure for doing this from assembly language and shows how to use the identification routine in programs written in Applesoft BASIC and Pascal.

The identification routine returns a value to the calling program that depends on the type of machine it is running on. Table 4-1 shows the return codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00</td>
<td>(0) = not an Apple IIe</td>
</tr>
<tr>
<td>$20</td>
<td>(32) = Apple IIe, but no Apple IIe 80-Column Text Card</td>
</tr>
<tr>
<td>$40</td>
<td>(64) = Apple IIe with 80-Column Text Card without auxiliary memory</td>
</tr>
<tr>
<td>$80</td>
<td>(128) = Apple IIe with Extended 80-Column Text Card</td>
</tr>
</tbody>
</table>
Note: An 80-column card installed in expansion slot 3 will work in an Apple IIe the same as in an Apple II or Apple II Plus, but it does not activate the built-in 80-column firmware. The identification program does not detect such a card, but returns a code of 32: no Apple IIe 80-Column Text Card.

Here is an outline of the procedure the identification routine uses to identify an Apple IIe and its variations:

1. Save four identification bytes from the ROM/RAM area ($0000 to $FFFF).
2. Disable interrupts.
3. Switch bank-switched memory to read ROM by reading $0089 twice.
4. Identify Apple IIe by finding the value 06 at $FB33.
5. If Apple IIe, and high bit is on at location $C017, then the computer has a text card.
6. If Apple IIe with 80-Column Text Card, then check for auxiliary memory:
   a. If $C013’s high bit is on, then reading auxiliary memory so must have auxiliary memory.
   b. If $C016’s high bit is on, then reading auxiliary zero page so must have auxiliary memory.
   c. If sparse memory mapping (no upper four address bits so that $800 has the same RAM location as $C00), then no auxiliary memory.
      1. Exchange a section of zero page with the section of code that switches memory banks. This way the zero page data is saved and the program doesn’t get switched out.
      2. Jump to the relocated code on page zero.
   3. Switch in auxiliary memory ($200 - $BFFF) for reading and writing by writing to $C005 and $C003.

Note: Auxiliary memory locations $400-$800 and $2000-$4000 may not be available depending upon the setting of soft switches for 80-column display and high-resolution graphics—they have priority over auxiliary memory selection.
4. Store a value at $800$ and see if same value at $\text{C}00$. If not, then auxiliary memory.

5. Change value at $\text{C}00$, and see if $800$ changes to same value. If so, then no auxiliary memory.

6. Set soft switches for reading and writing to main memory by writing to $\text{C}002$ and $\text{C}004$.

7. Jump back into program on main RAM.

8. Put zero page back.

7. Store identification byte for later reference by calling routine.

8. If Pascal routine then turn card back on by reading $\text{C}088$ twice.

9. The BASIC or assembly-language routines restore the RAM/ROM area as it originally was by checking four bytes saved at the start of the routine.

10. Enable interrupts.

11. Return to caller.

For some applications it may not be necessary to identify the exact configuration of the computer. For example, if your program cannot use the auxiliary memory, then you would not need to know whether it is available or not. In that case you may want to eliminate parts of the routine. For other applications the identification routine will use memory space required by your program, so you will need to move the routine to some other location.

**Warning**

If you change the identification routine, make sure that it still determines the configuration in the same way as the original. Later revisions of the Apple IIe may not support other identification procedures.

---

**Apple IIe Identification in Assembly Language**

The assembly-language subroutine given here is assembled to machine language in locations $200$ through $3CF$. To call the subroutine, your program does a jump to subroutine (JSR) to $204$. When the subroutine returns, the identification code is stored in memory location $3CF$.

---

**Programming Examples**
Apple IIe Identification Program

1982

; START OF CODE RELOCATED ON PAGE ZERO
; START OF FOUR BYTE LANGUAGE CARD 10
; DISABLE INTERRUPTS

SEI
LDA $E000
STA SAVE
STA $0000
STA SAVE+1
LDA $D400
STA SAVE+2
LDA $D800
STA SAVE+3
LDA $C081
LDA $C081
LDA $F883
CMP #66
BNE OUT1

; SAVE 4 BYTES FROM
; ROM RAM AREA FOR LATER
; RESTORING OF RAM RAM
; TO ORIGINAL CONDITION

IF NOT #6 THEN NOT APPLEII

; WAS 80 COLUMNS FOUND DURING STARTUP
; IF HT BIT ON THEN NO 80 COLUMN CARD
; SEE IF AUX MEMORY BEING READ
; AUX MEM BEING USED SO AUX MEM AVAIL.
; IF IF AUX ZP BEING USED
; AUX ZP BEING USED SO AUX MEM AVAIL
; NOT SURE YET SO KEEP CHECKING
; SWAP SECTION OF ZP WITH
; CODE NEEDING SAFE LOCATION DURING
; READ AUX MEM

MV

; JUMP TO SAFE GROUND
; BACK FROM SAFE GROUND, SAVE STATUS
; MOVE ZERO PAGE BACK

OUT4

; CARRY SET SO NO AUX MEM
; MADE IT SO THERE IS AUX MEM SET
; PARAM=800

OUT3

; 80 COLUMNS BUT NO AUX SO SET
; PARAM=540

OUT2

; APPLE IIIE BUT NO CARD SO SET
; PARAM=520

OUT1

; NOT AN APPLE IIIE SO SET PARAM=0
OUT    LDA $E000
      CMP SAVE
      BNE OUTON
      LDA $E000
      CMP SAVE+1
      BNE OUTON
      LDA $D400
      CMP SAVE+2
      BNE OUTON
      LDA $D800
      CMP SAVE+3
      BEQ G0OUT
    OUTON    LDA $C088
              ;NO MATCH, 50 TURN FIRST
              LDA $E000
              CMP SAVE
              BEQ OUTON0
              LDA $C080
              JMP G0OUT
    OUTON0    LDA $D000
                ;IF ALL LOCATIONS CHECK
                CMP SAVE+1
                BEQ OUTON1
                LDA $D080
                ;THEN DO NOTHING MORE
                JMP G0OUT
    OUTON1    LDA $D400
                ;IF ALL LOCATIONS CHECK
                CMP SAVE+1
                BEQ OUTON1
                LDA $D080
                ;OTHERWISE TURN ON BANK 2
                JMP G0OUT
    OUTON2    LDA $D0400
                ;CHECK SECOND BYTE IN BANK 1
                CMP SAVE+2
                BEQ OUTON2
                LDA $D080
                ;SELECT BANK 2
                JMP G0OUT
    G0OUT    LDA $C080
              ;SELECT BANK 2
              PLP
              ;RESET INTERRUPTS
              RTS

*** ROUTINE RUN IN SAFE AREA NOT AFFECTED BY MOVES ***

START    LDA $E00
          ;TRY STORING IN AUX MEM
          STA $C005
          ;WRITE TO AUX WHILE ON MAIN ZP
          STA $C003
          ;SET TO READ AUX RAM
          STA $D000
          ;CHECK FOR SPARSE MEM MAPPING
          LDA $C00
          ;SEE IF SPARSE MEMORY -SAME VALUE
          CMP $E00
          ;TK AWAY
          BNE AUXMEM
          ASL $C00
          ;MAY BE SPARSE MEM SO CHANGE VALUE
          LDA $D00
          ;& SEE WHAT HAPPENS
          CMP $C00
          BNE AUXMEM
          SEC
          BCS BACK

AUXMEM    CLD
          ;THERE IS AUX MEM
          BACK
          STA $C004
          ;SWITCH BACK TO WRITE MAIN RAM
          STA $C002
          ;SWITCH BACK MAIN RAM READ
          JMP ON
          ;CONTINUE PROGRAM ON PG 3 MAIN RAM
          DONE    NOP
          ;END OF RELOCATED PROGRAM MARKER
Apple IIe Identification from BASIC

One way to identify the configuration of an Apple IIe from BASIC is to load (using BLOAD) the machine-code version of the assembly-language routine described in the previous section, then execute a CALL statement to location 724 ($F04). When the subroutine returns to the BASIC program, executing a PEEK at location 975 ($3CF) gets the result.

Here is another approach to writing a BASIC program to identify the type of Apple II it is running on. In this program the assembled code for the assembly-language identification routine from the last section is included in the DATA statements.

```
10 DATA 8, 120, 173, 0, 224, 141, 208, 2, 173, 0, 208, 141, 209, 2, 173, 0,
   212, 141, 210, 2, 173, 0, 216, 141, 211, 2, 173, 129, 192, 173, 129,
   192, 173, 173, 251, 201, 6, 208, 73, 173
20 DATA 23, 192, 48, 60, 173, 19, 192, 48, 39, 173, 22, 192, 48, 34, 160, 42,
   190, 162, 3, 185, 0, 0, 150, 0, 153, 162, 3, 136, 208, 242, 76, 1, 0,
   8, 160, 42, 185, 162, 3, 153
30 DATA 0, 0, 136, 208, 247, 104, 176, 8, 169, 128, 141, 207, 3, 76, 73, 3,
   169, 64, 141, 207, 3, 76, 73, 3, 169, 32, 141, 207, 3, 76, 73, 3, 169,
   0, 141, 207, 3, 173, 0, 224
40 DATA 205, 208, 2, 208, 24, 173, 0, 208, 205, 209, 2, 208, 16, 173, 0, 212,
   205, 210, 2, 208, 8, 173, 0, 216, 205, 211, 2, 240, 36, 173, 136, 192,
   173, 0, 224, 205, 208, 3, 240, 6
50 DATA 173, 128, 192, 76, 169, 13, 173, 0, 208, 205, 209, 2, 240, 6, 173, 128,
   192, 76, 169, 13, 173, 0, 212, 205, 210, 2, 240, 6, 173, 128, 192, 76,
   169, 13, 173, 0, 216, 205, 211, 2
60 DATA 240, 3, 173, 128, 192, 40, 96, 169, 238, 141, 5, 192, 141, 3, 192,
   141, 0, 8, 173, 0, 12, 207, 238, 208, 14, 14, 0, 12, 173, 0, 8, 205, 0,
   12, 208, 3, 56, 176, 1, 2
70 DATA 141, 4, 192, 141, 2, 192, 76, 29, 3, 234
80 ALODE = 975: START = 724
90 FOR I = 0 TO 255
100 READ BYTE
110 POKE START + I, BYTE
120 NEXT
130 CALL START
140 RESULTS = PEEK (ALOKE)
150 PRINT RESULTS: REM RESULTS OF 0 MEAN NOT A II; 32 MEANS AIIIE BUT NO 80 COLUMN; 44 MEANS AIIIE WITH 80 COLUMNS BUT NO AUX MEM; 128 MEANS AIIIE WITH AUX MEM
160 END
```
Apple IIe Identification from Pascal

Here is the assembly-language identification program previously described in the form of a Pascal procedure.

```pascal
; MACRO POP ;SAVE PASCAL RETURN ADDRESS
PLA
STA %1
PLA
STA %1+1
; ENDM

; MACRO PULL_BIAS ;ADJUST FOR FUNCTION
PLA
PLA
PLA
PLA
; ENDM

; FUNC ID,0 RETURN .EQU 0 ;TEMP STORAGE OF RETURN TO PASCAL ADDRESS
SAFE .EQU 0002 ;START OF CODE RELOCATED ON PAGE ZERO

POP
PULL_BIAS
PHP
;LOCK OUT INTERRUPTS
SEI
LDA 0C0B9 ;ENSURE READING ROM BY TURNING OFF
LDA 0C0B9 ;BANKABLE MEM
LDA 0F883 ;SET APPLE IIe SIGNATURE BYTE
CMP #6
BNE OUT1 ;IF NOT #6 THEN NOT APPLE IIe
LDA 0C017 ;MAG 80 COLUMNS FOUND DURING STARTUP
BMI OUT2 ;IF HIGH BIT ON THEN NO 80-COLUMN CARD
LDA 0C013 ;SEE IF AUX MEMORY BEING READ
BMI OUT4 ;AUX MEM BEING USED SO AUX MEM AVAIL
LDA 0C016 ;SEE IF AUX ZP BEING USED
BMI OUT4 ;AUX ZP BEING USED SO AUX MEM AVAIL
LOD #2R ;NOT SURE YET SO KEEP CHECKING
MV LDX START-1,Y ;SWAP SECTION OF ZP WITH
STX SAFE-1,Y ;CODE NEEDING SAFE LOCATION DURING
LDA SAFE-1,Y ;READ AUX MEM
STA START-1,Y
DEY
```

Programming Examples
BNE MV
JMP SAFE
; JUMP TO SAFE GROUND
ON PHP
; BACK FROM SAFE GROUND, SAVE STATUS
LDY #2A
; MOVE ZERO PAGE BACK
MV2 LDA START-1,Y
STA SAFE-1,Y
DEY
BNE MV2
PLA
BCS OUT3
; CARRY SET SO NO AUX MEM
OUT4 LDA #80
; MADE IT SO THERE IS AUX MEM-SET
STA PARAM
; PARAM=180
JMP OUT
OUT3 LDA #40
; 80 COLUMNS BUT NO AUX SO SET
STA PARAM
; PARAM=180
JMP OUT
OUT2 LDA #20
; APPLE IIE BUT NO CARD SO SET
STA PARAM
; PARAM=120
JMP OUT
OUT1 LDA #0
; NOT AN APPLE IIE SO SET PARAM=0
STA PARAM
OUT LDA OCOR
; GET PASCAL BACK
LDA OCOR
; REACTIVATE INTERRUPTS
PLP
LDA 4D
; PUT 0 IN HIGH BYTE OF RESULTS
PHA
LDA PARAM
; PUT FOUND VALUE IN LOW BYTE & PUSH
PHA
LDA RETURN+1
; RESTORE PASCAL RETURN ADD
PHA
RTS

PARAM .BYTE
; ROUTINE RUN IN SAFE AREA NOT AFFECTED BY MOVES
START LDA #0EE
; TRY STORING _ IN AUX MEM
STA @005
; WRITE TO AUX WHILE ON MAIN ZP
STA @003
; SET TO READ AUX RAM
STA 0B00
; CHECK FOR SPARSE MEM MAPPING
LDA OC00
; SEE IF SPARSE MEMORY-SAME VALUE
JMP #0EE
; TK AWAY
BNE AUXMEM
ASL LDA 0C00
; MAY BE SPARSE MEM SO CHANGE VALUE
CMP OC00
; & SEE WHAT HAPPENS
BNE AUXMEM
SEC BACK
; SPARSE MAPPING SO NO AUX MEM
AUXMEM CLC
BACK
; THERE IS AUX MEM
STA #0004
; SWITCH BACK TO WRITE MAIN RAM
STA #0002
; SWITCH BACK MAIN RAM READ
JMP ON
; CONTINUE PROGRAM ON PG 3 MAIN RAM
DONE NOP
; END OF RELOCATED PROGRAM MARKER

Extended Text Card Supplement
Storing Graphics Pages from Applesoft

It is generally not practical to use the auxiliary memory from BASIC. A BASIC program can only move its variables in memory by getting very tricky with PEEK and POKE commands, an approach that is both inefficient and dangerous.

There is one form of data that uses lots of memory and is simple enough to handle from Applesoft: high-resolution graphics pages. The auxiliary memory is an ideal place to store as many as five complex graphics pages for rapid loading into the display buffer.

Like all of these examples, the following Applesoft example includes two short assembly-language subroutines. The first listing is the assembly-language form of the subroutines. The second listing is the Applesoft program with the machine-language subroutine included as a series of DATA statements. This method of adding a machine-language subroutine to a BASIC program is not very efficient, but it is convenient for short subroutines.

The program has two phases: in the first, the program generates five different high-resolution views and stores them in auxiliary memory; in the second, the program loads the stored graphics pages back into main memory one after another.

```
Hi-Res Page Mover for Auxiliary Memory Demo. Using AUXMOVE
Subroutine. July 1982
PARM = High byte of BUF. ADDR. (Page # times 32)
Call PUTPG to copy hi-res graphics page to AUX. MEM. location specified by PARM.
Call GETPG to load hi-res graphics page from AUX. MEM. location specified by PARM.

DSECT
ORG $3C
SRCBEG DS 2
SRCEND DS 2
DESTBEG DS 2
DEND

* PGBEG EQU $2000
* PGEND EQU $3FF8
* AUXMOVE EQU $3C11

* ORG $500
PARM DS 1
* MOVE HI-RES PAGE TO AUX MEM;
```
PUTPAG EQU * ; PAGE STARTING
LDA #PGTBEQ ; ADDRESS
STA SRCBEG
LDA #PGTBEQ+1
STA SRCBEG+1

LDA #PGTEND ; PAGE ENDING
STA SRCEND ; ADDRESS
LDA #PGTEND+1
STA SRCEND+1

* PARM = DESTINATION ADDRESS
* LDA #0 ; DESTINATION BEGINNING
STA DESTBEG ; ADDRESS
LDA PARM ; ADDRESS
STA DESTBEG+1

* USE AUXMOVE TO DO IT:
* SEC
JSR AUXMOVE
RTS

* COPY PAGE TO MAIN MEMORY
* GETPAG EQU * ; DESTINATION
LDA #PGTBEQ ; ADDRESS
STA DESTBEG
LDA #PGTBEQ+1
STA DESTBEG+1

* PARM = SOURCE ADDRESSES
* LDA #0 ; PARM FOR
STA SRCBEG ; ADDRESS
LDA PARM ; ADDRESS
STA SRCBEG+1

LDA #$F8 ; COMPUTE SOURCE
STA SRCEND ; ENDING ADDRESS
CLC
LDA PARM
ADC #$1F
STA SRCEND+1

* USE AUXMOVE TO DO IT:
* CLC
JSR AUXMOVE
RTS

Extended Text Card Supplement
Globe. Hi-res graphics demonstration for the Apple IIe Extended 80-Column Text Card.

This program draws five views of a rotating globe and stores five copies of the Hi-Res page in auxiliary memory. It then moves the views from auxiliary memory back into the Hi-Res graphics page in main memory, one after another. The rapid succession of views creates the impression of a solid rotating globe.

```plaintext
010 REM TEST: HOME
020 PRINT CHR$(17);: REM CTRL-Q for 40-column display
030 REM Pager subroutines in machine language:
040 DATA 161,0,133,60,169,0,133,32,133,61,169,248,133,32,133,62,169,63,133
050 DATA 65,169,0,133,66,173,0,3,133,67,56,32,17,195,96,0
060 DATA 161,0,133,60,169,0,133,32,133,61,169,248,133,32,133,62,169,63,133
070 DATA 65,169,0,133,66,173,0,3,133,67,56,32,17,195,96,0
080 REM Read the Pager subroutines and store at #301:
090 PARM = 788:PUTPAGE = 769:BRINGPAGE = 800
100 FOR I = 0 TO 64
110 READ BYTE
120 POKE PUTPAGE + I,BYTE
130 NEXT I
140 REM
150 REM Set up constants for drawing meridians (ellipses):
160 PI = 3.14159265: P2 = PI / 2
170 SP = P2 / 9: REM angle between meridians
180 EP = SP / 3: REM starting angle increment between views
190 DT = PI / 15: REM segment size (angle) for drawing meridians
200 B = 1: REM semi-major axis of ellipses.
210 REM Loop starting at 2000 draws five views and stores them:
220 FOR VIEW = 1 TO 5
230 HGR: HCOLOR = 3
240 DATA 60,0 TO 60,159 TO 219,159 TO 219,0 TO 60,0
250 VTAB 23: KTAB 9
260 PRINT "...constructing view #:"; VIEW
270 REM
280 IF DP = EP + VIEW THEN DIFFERENT starting angle each view.
290 REM Loop starting at 3000 draws meridians (ellipses):
300 FOR I = 1 TO 3000
310 IF A = COS (IANGLE) THEN SEMI-MINOR axis of ellipse.
320 IF FIRST = 1 THEN REM for plotting
330 IF FIRST = 0 THEN REM Loop starting at 4000 draws a meridian (ellipse):
340 FOR THETA = 0 TO PI STEP DT
350 LET X = A + SIN (THETA)
360 LET Y = B + COS (THETA)
370 NEXT THETA
380 NEXT I
390 NEXT VIEW
```

Programming Examples 45
Storing Data Strings from Pascal

These Pascal routines use locations $C00$ to $BFFF$ in the auxiliary memory for storage and retrieval of strings.

The code that moves the strings to and from auxiliary memory is stored at $E00$ in the Extended 80-Column Text Card. A separate initialization routine puts this code at $E00$, just once, to maintain system performance.

The retrieval routine is very fast, roughly equivalent to a MoveLeft from a Packed Array of Char. The storage routine is less efficient; if speed is important in your program, you may want to try to optimize it.

Like the other examples, these routines were written for a particular application and are not general-purpose subroutines. They are included here to show you the kind of thing you can do with the auxiliary memory.
The following routine is performed only once. The routines that move strings in and out of the Extended 80-Column Text Card are moved to E000 in the auxiliary memory.

```
.TITLE "ASSEMBLY ROUTINES 1IE INITIALIZATION"
.PAGE
.NOMACROLIST
.NOPATCHLIST

RMAIN48 .EQU DC002 ; SOFT SWITCHES, SEE
RDAUX48 .EQU DC003 ; THE REFERENCE MANUAL
WRMAIN48 .EQU DC004
WRAUX48 .EQU DC005
RMAIN16 .EQU DC006
WRAUX16 .EQU DC009
HIRESOFF .EQU DC056

RETURN0 .EQU 028
RETURN1 .EQU 02A

; REGISTER MAP
ZREG00 .EQU 0
ZREG02 .EQU 4
ZREG04 .EQU 6

OUT4 STA RWAUX16 ; WRITE AUX MEMORY
LOY #80. ; LENGTH OF PATCH
OUT4MEM LOA E001STUF-1,Y ;
STA 0E001,Y
DEY
BNE OUT4MEM

OUT4MEM1 LOA E02STUF-1,Y ; CODE NEEDING SAFE LOCATION
STA 0E101,Y
DEY
BNE OUT4MEM1

STA RMAIN16 ; WRITE MAIN MEMORY
STA HIRESOFF ; MAKE HIRES P. AVAILABLE

; END OF THIS ROUTINE

Purpose: Moves a string from auxiliary memory to Pascal.

If the program finds the Extended 80-Column Text Card, the following code is moved to E002.

The program gets here from a JSR in MOVE_FR_AUX, and goes back so that the auxiliary memory can be turned back off. Zero page on the extended text card contains ZREG00 and ZREG02; they are the arguments for the move. Stack usage: The return address in 48K main memory is stored in the auxiliary stack. This is the only use of the auxiliary memory stack.

Programming Examples
Extended Text Card Supplement
; SPACE IS FULL, SO RETURN ZERO
LDA #0
TAY
STA (ZREG02),Y ; RETURN A ZERO, FULL
INY
STA (ZREG02),Y
BNE E102FAIL ; UNCONDITIONAL

; THERE IS STILL ROOM, SO CONTINUE
E102C1
LDY #1
STA (ZREG02),Y ; STORE IN RETURN ADDR
DEY
STA ZREG04+1 ; SETUP THE MOVE
LDA NEXTAVAIL
STA (ZREG02),Y ; LOW BYTE OF RETURN
STA ZREG04 ; MORE OF THE MOVE

; NOW INCREMENT THE NEXT AVAILABLE ADDRESS
CLC
ADC #1 ; ADD 1 FOR STRING LENGTH
BNE +5
INC NEXTAVAIL+1 ; ROLLED INTO NEXT PAGE
CLC
LDY #0
ADC (ZREG00),Y ; ADD LENGTH OF STRING
STA NEXTAVAIL
BCC +5
INC NEXTAVAIL+1 ; INTO NEXT PAGE
STA WRAX48 ; WRITING INTO AUX 48K
LDA (ZREG03),Y ; READING AUX 48K USING AUX ZERO PAGE
STA (ZREG04),Y
BNE E102EXIT
TAY
E102LOOP
LDA (ZREG00),Y
STA (ZREG04),Y
DEY
BNE E102LOOP
E102EXIT
STA WMAIN48 ; NOW WRITE MAIN MEM
RTS ; GOING BACK TO 48K RAM
.E00

; The following code is linked into the main Pascal program. This code
; stores the arguments in the auxiliary zero page and then jumps to E00 on
; the Extended 80-Column Text Card.

; TITLE "ASSEMBLY ROUTINES FOR ITE"
; PAGE
; NODROP
; NOPATCH

; RMAIN48 .EQU 0002
; RAX048 .EQU 0003
; WMAIN48 .EQU 0004
; WRAX48 .EQU 0005
; RWMAIN16 .EQU 0009
; RWMAIN18 .EQU 0008

; RETURN ADDRESS ZERO PAGE LOCATIONS

; RETN0 .EQU 028
; RETN1 .EQU 02A

Programming Examples
REGISTER MAP

ZREG00 .EQU 0
ZREG02 .EQU 4

.TITLE "MOVE STRINGS FROM I to AUXILIARY MEMORY"
.PROC

PROCEDURE MOVE_FROM_AUX (FROMA; VAR TOA) (* Move string *)

Purpose: Move a string from auxiliary memory to Pascal. Most of the actual move is done at auxiliary memory location E002.

Stack usage: Input, output addresses.

; STORE RETURN ADDR IN AUX ZERO PAGE
; POP RETURN0 ; RETURN TO PASCAL
; ADDRESSES ARE TWO BYTES. PULL BOTH BYTES OFF THE MAIN STACK, THEN SWITCH
; TO AUX ZERO PAGE AND STORE BOTH BYTES.
PLA
PLA
STA RWAUX16
STX ZREG02
STA ZREG02+1
STA RWM3N16
; STORE FROM ADDRESS IN AUX ZERO PAGE
PLA
PLA
STA RWAUX16
STX ZREG00
STA ZREG00+1
; NOW GET OVER TO AUX PAGE AND DO IT ALL
JSR 0E002
; NOW PROCESS COMING BACK FROM E002 IN AUX MEMORY
STA RWM3N16
PUSH RETURN0
RTS

.TITLE "MOVE STRINGS TO I from AUXILIARY MEMORY"
.PROC

PROCEDURE MOVE_TO_AUX (VAR FROMA; VAR TOA) (* Move string *)

Purpose: Move a Pascal string to auxiliary memory. Most of the actual move is done at auxiliary memory location E102.

Stack usage: Input, output addresses.

; STORE RETURN ADDR IN AUX ZERO PAGE
; POP RETURN0 ; MAIN ZP STACK
; STORE TO ADDRESS IN AUX ZERO PAGE
  PLA
  TAX
  PLA
  STA   RWAUX16
  STX   ZR502
  STA   ZR502+1
  STA   RWMAIN16
; STORE FROM ADDRESS IN AUX ZERO PAGE
  PLA
  TAX
  PLA
  STA   RWAUX16
  STX   ZR500
  STA   ZR500+1
; NOW GET OVER TO AUX PAGE AND DO IT ALL
  JSR   OL04
; RETURN FROM E104 IN AUX MEMORY
  STA   RWMAIN16
  PUSH   RETURN0
; .END

; LSB OF ADDR TO RETURN
; MSB OF ADDR TO RETURN
; SWITCH TO AUX ZP
; IN AUX ZERO PAGE
; STILL IN AUX MEM
; SWITCH TO MAIN ZP
; LSB OF INPUT STRING
; SWITCH TO AUX ZP
; IN AUX ZERO PAGE
; STILL IN AUX MEM
; JUMP OVER NEXTAVAJ AT E102
; MAIN TO MAIN ZP AND TOP

Programming Examples
A
accumulator 30, 31
address bus 7
address 30
data 29
destination starting 30
program starting 31
source starting 30
source ending 30
transfer 31
alternate zero page 25
ALTZP soft switch 25
figure 27
table 26
warning 29
Annunciator 3 11
display page used 12
Apple II 35, 36
compatibility 25
system software 15
Apple II Plus 35, 36
compatibility 25
Apple Ile viii, 7, 35
control keys 4
display memory 18
display modes 4
display pages 16
double high-resolution graphics 11-12
double high-resolution graphics 11-12
escape sequences 4
identification from assembly language 37
identification from BASIC 40
identification from Pascal 41
installing card in 3
processor 4
Rev A 12
Rev B 11
Apple Ile Identification Program 38-39
Apple Ile Identification Program (Pascal) 41-42

Applesoft BASIC 17, 35
program example using AUXMOVE 44
using to store graphics pages 43
application programs viii, 21, 35
running with the card vii
assembly language 35, 37
subroutines 15, 37, 43
programs to use auxiliary memory 29
routine for Pascal 41
AUX. CONNECTOR 3
See also auxiliary slot
auxiliary memory viii, 4, 7, 9,
  10, 11, 12, 15, 16, 18, 19, 21,
  22, 25, 29, 31, 36, 37
accessing 4
address bus 9
addressing 7
amount 3
careless switching 4
how it works 7-12
select switches, table 26
subroutines 29-31
switching to from bank-switched memory 9
transferring control to 30-31
using viii
using from a program, examples, 35-51
warning 9, 15
Auxiliary Memory String Routines 47-51
auxiliary RAM 15
auxiliary slot 11
AUX. CONNECTOR 3
AUXMOVE subroutine 15, 29, 30
eexample using (Applesoft) 44
parameters 30
warning 29
B

bandwidth, video monitor 11
bank switches 9
bank-switched memory 9, 25, 36
warning 9
with auxiliary memory 9
BASIC 15, 37, 40, 43
Applesoft 17
Integer 17
reading soft switches from 17
warning using 15
bit 0 29
bit 7 17
bit
carry 29, 31
color-select 19
high-order 19, 22, 25
overflow 31
sign 17, 22, 25
built-in 80-column display routines 16
built-in I/O routines 21
built-in subroutines 29, 30
bus
address 7
data 7
byte
high-order 30, 31
low-order 30, 31

C

CALL statement 15, 40
carry bit 29, 30, 31
circuitry, display 18
CLC instruction 29, 30
clock cycle 9
with 80-column display on 10
CLV instruction 31
color monitor, RGB 10, 11
color-select bit 19
commands
PEEK 17, 43
POKE 17, 43
computer configuration, identifying 35-37
configuration 35
configuration of the computer 35-37
control keys 4
control switches 22

D

data addresses 29
data bus 7, 11
data, moving 29
DATA statements 40, 43
data strings 46
decimal 17
destination starting address 30
display buffer 43
display circuitry 18
display cycle 9
display data 10, 21
display modes 4
display pages 16
control switches 22
length 9
locations, table 16
See also graphics pages
display soft switches, table 18
display, 80-column 19
figure 20
DOS 3.3 viii, 15, 21
dot patterns 10
double high-resolution graphics (560 dot) 11-12, 19
obtaining 11
board necessary for 12
with Rev A Apple Ile 12

E

80-column card, any 36
80-column display 10, 11, 16, 19, 36
addressing directly 18
bandwidth needed 10
firmware 3, 4, 36
figure 20
map 10
storing data 9
switch for 11
with clock cycles 9
versus 40-column display (dot patterns) 10
SO COL soft switch 18
80-Column Text Card 4, 16, 35, 36
80-column text 11, 12, 19
mode 12
STO RE soft switch 18, 19, 22
figure 23, 24, 28
table 26
warning 21

Extended Text Card Supplement
escape sequences 4
expansion slot 3 36
Extended 80-Column Text Card 11, 35, 46
buffer 10
connecting pins 11
differences from 80-Column Text Card viii
storing data directly 19

F
firmware, 80-column display 3, 4, 36
560-dot graphics 11, 12, 16
See also 280-dot graphics
40-column text 11
mode 16
48K bank, switching 21-22
48K memory space 9, 21
14MHz 10, 11
functions
keyboard 17, 25
Pascal 15

G
Globe 45-46
graphics mode, double high-resolution 12, 19
graphics Page 1X (high-resolution) 19
graphics pages 43
storing from Applesoft 43
See also display pages
graphics
280-dot 16
560-dot 12, 16
double-high-resolution 19
double-high-resolution memory mapping 19
high-resolution, Page 1 19, 22
high-resolution, Page 1X 19
high-resolution program example 45-46

H
hexadecimal 16, 17
high-level languages 15
high-order bit 17, 19, 22, 25
high-order byte 30, 31
high-resolution graphics 11, 36, 43
Page 1 12, 19, 22
Page 1X 19
pages 43
program example 45-46
switch for 11
HIRES soft switch 18, 19, 22
figure 24, 28
table 26
warning 21
horizontal resolution 11

I
identification return codes, table 35
identification routine 35, 40
outline 36-37
warning 37
identification, Apple IIe (assembly language) 37
identification, Apple IIe from BASIC 40
installation 3
instruction
CLC 29, 30
CLV 31
JSR 37
jump 31
SEC 29, 30
Integer BASIC 17
interpreter, warning using 15
interrupts 36, 37
I/O 7
links, standard 21
routines, built-in 21
subroutines 29

J
JSR instruction 37
jump instruction 31
jump to subroutine (JSR) 37

K
keyboard functions 17, 25

L
low bandwidth monitor 19
low-order byte 30, 31
low-resolution graphics 11, 16

M
machine language 17
programs 17
main logic board
buffer 10
Rev A 12
Rev B 11
main memory viii, 7, 9, 10, 12, 15, 16, 18, 19, 21, 25, 31
address bus 9
switching 4
memory
amount card contains viii auxiliary 15
select switches, table 26
bank-switched 25
reading 21, 25
writing 21, 22, 25
memory locations 17
Memory Management Unit 7
memory map 9, 12, 19
diagram 8
memory pages, length 9
memory switching 15
mixed graphics modes 11
MIXED soft switch 18
MMU 7
modes, mixed graphics 11
Molex-type pins 11
Monitor program 19
monitor, low bandwidth 19
monochrome monitor 11
moving data 29
N
negative decimal 17
O
operation
read 17, 25
warning with 17
write 17, 25
warning with 17
overflow bit 31
P
Page 1
display 19
graphics 22
high-resolution graphics 12, 19
text 16, 18, 22
page one 9
warning 29
Page 1X
display 16, 18, 19
high-resolution graphics 19
text 16
pages, high-resolution
graphics 43
Page 2, text 16
PAGE 2 soft switch 18, 19, 22
figure 28
table 26
warning 21
page zero 9, 29, 30, 31
warning 29
parameters 30
Pascal 30, 35, 37, 41
1.1 viii, 15
functions 15
procedures 15, 30
routines, programming
elements 47-51
storing data strings from 46
warning using 15
PEEK 17, 40, 43
POKE 43, 17
power-on light 3
procedures, Pascal 15, 30
procedures, assembly-language 29
processor 4
processor status word 29, 31
program starting address 31
program
Apple Ile identification 38-39
Apple Ile Identification
(Pascal) 41-42
Applesoft example using
AUXMOVE 44
assembly-language 29
machine-language 17
Monitor 19
programmable memory,
amount 4
programming examples 35-51
R
RAM 4, 7
amount 3
auxiliary memory 15
RAMRD soft switch 21, 22
figure 23, 24
table 26
RAMWRD soft switch 21, 22
figure 23, 24
table 26
read operation 17, 25
warning 17
reading memory 21, 25
registers, X and Y 30
Rev A Apple Ile 12
Rev A main logic board,
warning 12
Rev B main logic board 11
RGB color monitor 10, 11
ROM  7, 21
routine, identification  35, 40
outline  36-37
routines, built-in  30

S
SEC instruction  29, 30
7MHz  10
sign bit  17, 22, 25
6502 microprocessor  7
6502 stack  9
warning  29
slot  3, 36
soft switches  7, 11, 15, 16, 17,
29, 31, 37
accommodating  25
ALTZP  25
figure  27
table  26
warning  29
Annunciator  3, 11
80COL  18
80STORE  18, 19, 22
warning  21
figure  23, 24, 28
table  26
HIRES  18, 19, 22
warning  21
figure  24, 28
table  26
MIXED  18
PAGE2  18, 19, 22
figure  28
table  26
warning  21
RAMRD  21, 22
figure  23, 24
table  26
RAMWRT  21, 22
figure  23, 24
table  26
setting  15, 31
TEXT  18
warning  17, 21
software, Apple II system  15
source ending address  30
source starting address  30
sparse memory mapping  36
stack  25, 31
warning when switching  15
stack pointer, warning  31
standard I/O links  21
string routines, Pascal
examples  47-51
subroutine call  31

subroutines
assembly-language  15, 37,
43
auxiliary-memory  15, 29-31
AUXMOVE  15, 29, 30
parameters  30
warning  29
built-in  29
I/O  29
XFER  15, 29
parameters  31
warning  31
switch locations  17, 21
decimal  17
hexadecimal  17
negative decimal  17
switches, control
48K bank  22
display-page  22
system software for the
Apple II  15

T
television set  10, 19
text display, 80-column  12, 19
figure  20
text Page 1  9, 16, 18, 22
text Page 1X  16
text Page 2  16
TEXT soft switch  18
transfer address  31
280-dot graphics  11, 16
See also 560-dot graphics

V
video display page locations,
table  16
video displays  16
video monitor  10
video display generation  11

W
write operation  17, 25
warning  17
writing memory  21, 25

X
X and Y registers  30
XFER subroutine  15, 29, 30
parameters  31
warning  31

Z
zero page  25, 29, 36, 37
warning when switching  15